LA-UR -83-3444

[illegible] 9

LA-UR--83-3444

DE84 003017

TITLE: PARTICLE ACCELERATOR CONTROL AND DATA ACQUISITION IN THE CONTEXT OF VAX/VMS

AUTHOR(S): Stuart C. Schaller and Jamil K. Corley

SUBMITTED TO: FALL DECUS U.S. Symposium, October 23 thru 28 1983

PARTICLE ACCELERATOR CONTROL AND DATA ACQUISITION IN THE CONTEXT OF VAX/VMS*

Stuart C. Schaller and Jamil K. Corley
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

The Los Alamos Meson Physics Facility (LAMPF) Control System monitors and controls a linear accelerator through more than 10,000 widely disparate I/O devices. The heart of the Control System software is the Data System, which provides a uniform application program interface based on symbolic device names. In many ways the Data System parallels the VAX/VMS Record Management Services (RMS) in its needs for asynchronous operations, protection, and locks for multiprocess interactions. Since the accelerator control hardware is continually changing, it is important that privileged code be kept to a minimum or be testable in a non-privileged environment.

This paper describes the LAMPF Data System design including the use of VAX/VMS user written system services (both kernel and supervisor mode), a user supplied image rundown routine, the VAX/VMS lock manager, and a large (3.5 Mbyte) protected global section.

INTRODUCTION

The Los Alamos Meson Physics Facility (LAMPF) at Los Alamos National Laboratory is a linear proton accelerator capable of accelerating hydrogen ions to an energy of 800 MEV. The H+ injector can produce one milliamp beams of protons. Two other injectors produce less intense beams of H- and polarized H- ions. The linear accelerator (linac) accelerates one to two millisecond bursts of ions 120 times a second.

At the end of the linac, the beams pass through a switchyard containing magnets which direct them to the appropriate experimental areas. In the experimental areas, the nucleons are used directly for nuclear physics research, or they are used to produce copious quantities of pi- and mu-mesons which are themselves collimated into secondary beams. High intensity bursts of neutrons can also be produced for neutron physics work, and, at the main beam stop, remaining particles are used for neutrino physics experiments and radio-isotope production.

The LAMPF Control System (LCS) is responsible for controlling the injectors, linac, switchyard, and primary beam lines (up to the pion/neutron production targets and primary beam stops). More than 10,000 data acquisition and control devices are located along the one kilometer length between the injectors and the beam stops. The LCS Data System provides application programs (i.e., those programs which actually deal with the physics and operation of the accelerator) with a uniform interface to the devices that control and acquire data from the accelerator.

This paper describes the LCS Data System design and implementation. The first half of the paper deals with the LAMPF Control System context and the Data System design goals and requirements. The second half deals with the implementation of the Data System, paying particular attention to the use of the VAX/VMS operating system.

*Work performed under the auspices of US Department of Energy

THE LAMPF CONTROL SYSTEM

History

The original LAMPF Control System was written for a Systems Engineering Laboratory (SEL) 840 computer. Locally designed Remote Instrumentation and Control Equipment (RICE) hardware provided relatively high speed access to accelerator data and controls. Eventually, additional accelerator data at higher data acquisition rates were made available through CAMAC (IEEE Std 583 - 1975) interfaces addressed by remote PDP-11 computers. The remote PDP-11s were linked to the SEL-840 via locally designed CAMAC data link modules.

Several years ago, a decision was made to replace the SEL-840 with a VAX-11/780. The older computer was becoming difficult to maintain and the control system software difficult to modify. An interesting discussion of the problems of a production environment control computer upgrade may be found in [1].

Current System Configuration

The new VAX-based LAMPF Control System hardware organization is shown in Figure 1. This is the configuration the LCS Data System must handle. The VAX-11/780 - used as the central control computer - is at the hub of a star network. All application programs which communicate with the accelerator operators run on this computer.

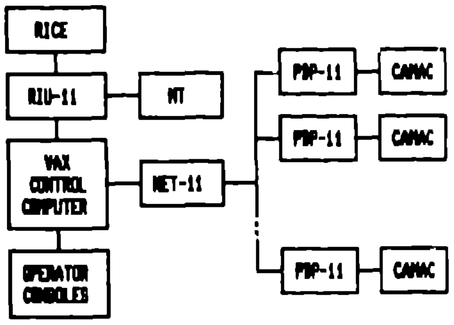

Figure 1 - LAMPF Control System Context

The RIU-11 is a PDP-11/34 computer used to dual-port the RICE system to the VAX and the SEL computers during the upgrade process. The RIU-11 accepts messages from the control computer and causes RICE data takes and commands to be issued through the RIU (RICE Interface Unit). The RIU-11 also receives timing signals and gate information from the LAMPF Master Timer (MT). These signals allow the RIU-11 to make timed and untimed RICE data take requests with specific timing gate configurations.

The NET-11 is a PDP-11/34 which is used to dual-port the remote PDP-11s during the upgrade. It switches messages between the control computer and the remote PDP-11s. The remote PDP-11s contain software which allows the control computer to directly address CAMAC devices in the associated CAMAC crates. In addition, the remotes also contain software for extremely time-dependent data acquisition tasks (such as acquiring beam profiles).

Note that the VAX control computer does not do real-time data acquisition. It only needs to respond to operator requests in human response times. All beam-pulse-related, real-time processing occurs in the RIU-11 or in the remote PDP-11s connected to the NET-11. Also note that the VAX, the RIU-11, and the NET-11 are located in close proximity. The RICE hardware and the remote PDP-11s are distributed throughout the accelerator and experimental areas.

DATA SYSTEM DESIGN

Goals and Requirements

At the time the VAX upgrade was started, the intention was to rewrite the application software which existed on the SEL-840 control computer, but not to modify software on other computers in the control system. The LCS Data System was proposed to help simplify this immense task. The primary goal was to introduce the Data System as an interface between the VAX applications and QIO-based message exchanges between the control computer and other LCS data acquisition hardware.

We wanted the Data System to be a robust software system that was flexible and maintainable, and that allowed a simple yet powerful application program interface. To this end we wanted the application programs to be as independent of the hardware as possible. In particular, we wished to prohibit the use of hardware addresses by application programs.

The hardware that already existed (and the software in the case of the remote computers) dictated requirements for detailed Data System design. Since several different hardware protocols were in use, the Data System had to interpret the application program read and command requests and perform the appropriate hardware actions. This included cases where a single request could result in a number of hardware actions, i.e., to establish set points or average data. In addition, the Data System had to allow default or explicit specification of a multitude of request parameters. These parameters included data take tim'ng, command protection, request synchronization, and optional conversion between raw and engineering units.

Since LAMPF is a production facility operating 24 hours a day, the Data System had to be protected against accidental modification, and it had to have a rich set of diagnostic tools for troubleshooting and development. We therefore wanted the ability to monitor Data System activity and the ability to simulate Data System devices during application program development. We wanted the monitoring and simulation capabilities to be available on a system-wide, device-specific, or process-specific basis.

Hardware Independence

A symbolic device-naming scheme using 10-character "operator designators" which had been in use at LAMPF for RICE devices was extended to cover all Data System devices. We insisted that all references to accelerator devices by application programs be by operator designator. We also made the operator designators reflect the actual physics of the accelerator as opposed to reflecting accidents of hardware address assignments.

To implement the universal use of operator designators, we needed a data base describing the devices and a Data System interface allowing application programs to make read and command requests via operator designators.

The LCS "Device Tables" define the Data System devices available to the application programs and provide the mapping from operator designators to hardware information. The Device Tables, maintained by a commercial data base management system, provide a central location for all information about accelerator devices. More information concerning the LCS Data System Device Tables may be found in [2].

Each operator designator has an associated Device Table Entry. A Device Table Entry contains two kinds of information about a device. The first kind is device characteristic information including the domain (digital or analog), the dimension (scalar or vector), allowable requests, the equipment type, and the hardware address. The equipment type can be: RICE - for devices in the RICE system; CAMAC - for directly controllable CAMAC modules; Remote - for devices defined by software in the remote PDP-11s; or Pseudo - for devices defined by software in the VAX control computer.

The second kind of Device Table Entry information describes the default parameters to be used for read or command requests. If a program requests a read on a device and specifies no additional parameters, the read request will be defined by the defaults contained in the associated Device Table Entry. Default request information includes: a delay gate; timing parameters allowing timing relative to a specific edge of a specific Master Timer gate; and required beam Master Timer gate configurations.

DATA SYSTEM IMPLEMENTATION

Languages and Libraries

The language used to implement the Data System was the standard language used throughout the LAMPF control system, FORTRAN. The limited FORTRAN control structures were augmented by using a FORTRAN preprocessor called FLECS.

The Data System is a set of routines residing in a sharable runtime library. We choose this method of implementation for two reasons. First, it was important that a minor change not require relinking the entire base of application programs -- a process that could involve as many as two thousand programs. Second, we needed privileged code, so included in this library were user written system services which allowed the privileged Data System operations to be performed without the application programs needing special privileges.

Data System Global Section

We realized that we needed several global data bases: for the Device Table device descriptions, for Master Timer state information, and for global information about the Data System and the state of current application program requests. We decided to place these data bases in a single VAX/VMS global section. The Data System global section is mapped to each program using the Data System. Thus references to information in the global section are handled by the VAX/VMS paging mechanism.

Originally we had planned to access the Device Table data base at run-time with a commercial database retrieval system. Prototype versions of the Data System showed that this was simply too slow and that no other data base management system was likely to be significantly faster. We decided to keep the commercial system for maintaining the device tables, and we wrote a program which reads the Device Tables and builds the global section containing the device descriptions. Run-time access to a given device is provided via an operator designator based hash table.

For diagnostic purposes we wanted to have global information on the state of the Data System, processes using the Data System, devices currently open, and requests in progress. Hence we set aside a portion of the global section for dynamic allocation of blocks of storage describing open devices, pending operations, and message buffers. The dynamic block allocation/deallocation routines are one of the few areas of the Data System where it is necessary to go to kernel mode in order to raise IPL to ASTDEL for preventing process deletion. Our primary tool for synchronizing access to the dynamic portion of the global section is the VMS lock manager.

To prevent application programs from inadvertently modifying information in the global section, the section is protected (user read/supervisor write). Note that we require a mechanism to clean up the global section if an application program terminates abnormally.

Application Program/Data System Interaction

Figure 2 shows how a typical application program interacts with the Data System. An application program that uses the Data System is linked with the LCS run time library. The Data System global section is mapped to the program's virtual address space when the Data System is assigned. The Data System routines then access device descriptions in the global section and communicate with various I/O drivers to service the program's read and command requests.

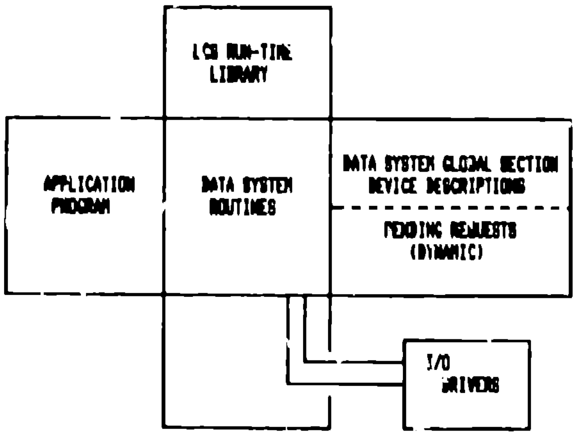

Figure 2 - Application program using the Data System

In some instances of a Data System request there are a series of operations that must be performed before the request is completed. This requirement coupled with the desire to give the user asynchronous capability prompted our decision to make the Data System routines AST driven. The completion of each QIO to the various I/O drivers is signaled via an AST. If it is requested, the Data System interface will declare an AST for the application program.

User Written System Services

Most of the Data System user written system services are in supervisor mode for three major reasons.

First, we wanted to protect the Data System global section from accidental corruption. We were willing to grant read access to any process, but we needed to control write access. An elevated mode of access seemed the most feasible mechanism for the Data System to use to update the global section.

Second, when the control computer is controlling the beam, we cannot take the machine for system development time. We used supervisor mode as our elevated access mode to minimize the impact of development and bug tracking. The advantages of supervisor mode are that an exception in this mode won't bugcheck the system and is less likely to damage VMS.

Finally, there are instances in the Data System request processing when one user request translates into several I/O operations. We did not want the Data System to be blocked by a user's ASTs, or by a user being at AST level. We therefore chose to do our I/O operations in supervisor mode so that the Data System ASTs would be delivered in supervisor mode.

Some of our system services run in kernel mode. These include the allocation and deallocation routines for the global section dynamic region, routines to set and unset privileges, and the routines entered via the user rundown vector.

Simulation and Logging

The Data System allows both simulation and logging to be enabled or disabled on system-wide, process-specific, and device-specific bases. When logging is enabled, information concerning selected Data System actions is sent to a system-wide Data System log mailbox. The Data System log process saves the log entries on a disk file which can be interrogated either on-line or off-line.

Simulation occurs in a similar fashion. If a user's request requires simulation, the action level messages are sent to the IMPF (Imaginary Meson Physics Facility) process instead of an I/O driver. The IMPF process simulates LAMPF devices on the hardware level and returns an appropriate response to the Data System Interface.

VAX/VMS EXPERIENCE

In many ways our implementation parallels the VAX/VMS Record Management Services (RMS), since our requirements are similar.

We require device independence, and asynchronous operations. We also need multi-process synchronization and protection against accidental interference with I/O sequences and unauthorized I/O operations. In the case of RMS, synchronization and protection maintain the integerity of the file system. In our case they prevent accidental device movement, which could cause beam spill or device damage.

The Data System runs in elevated mode to protect data structures. The user rundown capabilities of VMS allow the Data System to complete or cancel pending I/O at image rundown. In addition, the Data System uses the VAX/VMS lock manager to coordinate accesses to queues.

Our operation structure also parallels RMS. The Data System linkage is set up with an assign. Each device must be opened. Any series of requests may be issued, in the form of reads, and commands. These requests are translated by the Data System into one or more actions to be performed by the appropriate driver or process. Notification of completion comes to the Data System routines via ASTs, and is passed on to the user program. At image rundown, all connections are cleaned up via the Data System rundown routine.

One of the more interesting aspects of the project was running in supervisor mode.

Since there is no VMS supplied change mode to supervisor system service, we wrote a kernel mode system service which enters supervisor mode using a mechanism similar to the one used by VMS for AST delivery.

We were not able to set breakpoints in supervisor mode using the standard debugger. An exception in supervisor mode terminated the process, not just the image. We eventually wrote a supervisor mode condition handler to give supervisor mode exception tracebacks. For easier debugging we developed a scheme where we could run code in either supervisor mode or user mode by setting a switch. This mechanism allowed us to use the production global section or smaller test sections.

We assigned a number of logical symbols at system level, and wrote a command procedure that asked the developer a number of questions including the mode setting, global section, and set of diagnostic tools to be used. The development system has its own library, which is a copy of the runtime library routines with the addition of a transfer vector section. The transfer vector for the Data System assign routine translates the logical symbols and stores the global section name and mode of operation. Whenever another privileged routine is called, the transfer vector for this routine checks its call mode. This mechanism has proved invaluable in code development, although there are certain timing problems that are masked or created by running in user mode.

Using FORTRAN as our major language has been a problem because it is not AST re-entrant. In some cases we were forced to write small routines in MACRO. In other cases it was feasible to disable ASTs for a short period of time.

CONCLUSIONS

We have been able to take data and control RICE devices through the new Data System. Application programs have been written for the VAX. The Data System interface exists in the LCS run time library and maps to the 3.5 Mbyte Data System global section.

We feel we have achieved our design goals of flexibility and hardware independence. We have much left to implement in the way of new devices and expanded simulation and diagnostic capabilities, but our experience indicates that these extensions will fit in well with our design.

The VAX/VMS operating system has provided a flexible environment in which to work at a high level with protection and synchonization mechanisms. In particular, running in supervisor mode has afforded the protection we needed without interfering too drastically with VMS operation. The AST driven routine structure has proven a useful and flexible mechanism for the over-all flow of control of the Data System, while the availability of a user written rundown has proved valuable in cleaning up the application program's requests. Use of a large global section for the necessary data bases has speeded up the process significantly. The ability to debug new privileged code in a user mode environment has proved invaluable in adding new Data System capabilities.

ACKNOWLEDGMENTS

We would like to thank Eric Bjorklund, Patricia Rose, Phyllis Wallis, and Stanley Brown for their contributions to the Data System design and implementation.

REFERENCES

[1] Schultz, David E., and Brown, Stanley K., "On-Line Replacement of a Particle Accelerator Control Computer," Proc. IEEE Real-Time Symposium, Dec. 1981, pp.78-82.

[2] Brown, Stanley K., "The Use of a Commercial Data Base Management System in the LAMPF Control System," IEEE Trans. on Nuclear Science, Aug. 1983, Vol. NS-30, No. 4, pp. 2302-2304.

[3] Schaller, Stuart C., and Rose, Patricia A., "Data Acquisition Software for the LAMPF Control System," IEEE Trans. on Nuclear Science, Aug. 1983, Vol. NS-30, No. 4, pp. 2308-2310.